PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2003-288129
(43)Date of publication of application : 10.10.2003

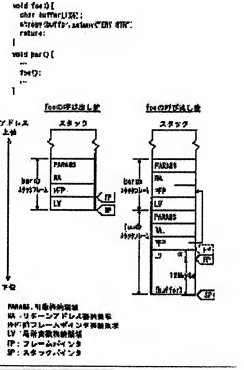
(51)Int.Cl. G06F 1/00
G06F 9/42
G06F 9/45
G06F 12/14

(21)Application number : 2002-092219
(22)Date of filing : 28.03.2002

(71)Applicant : SUN ATMARK:KK
(72)Inventor : HARIKAWA YORIYUKI

(54) MEMORY MANAGEMENT METHOD, MEMORY DEVICE, COMPUTER SYSTEM, COMPILER AND PROGRAM

(57)Abstract:
PROBLEM TO BE SOLVED: To provide a memory management technique for avoiding stack smashing attack.
SOLUTION: When securing a local variable storing area LV for a stack of a subroutine, the size of the local variable storing area LV is determined randomly in a range larger than the required size for storing all local variables declared, in a source program of the subroutine. Thus, because it is impossible to precisely anticipate the relative position of a return address storing area RA, in a stack frame from the source code or the like, it becomes very difficult to rewrite the return address with a destination address (the start address of the malicious program code) by the stack-smashing attack.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（A）**　(11)特許出願公開番号
特開2003－288129
（P2003－288129A）
(43)公開日　平成15年10月10日(2003.10.10)

(51)Int.Cl.⁷	識別記号	ＦＩ		テーマコード*(参考)
Ｇ０６Ｆ　1/00		Ｇ０６Ｆ　9/42	３３０Ａ	５Ｂ０１７
9/42	３３０	12/14	３２０Ａ	５Ｂ０３３
9/45		9/06	６６０Ｊ	５Ｂ０７６
12/14	３２０	9/44	３２２Ｈ	５Ｂ０８１

審査請求　未請求　請求項の数５　ＯＬ　（全５頁）

(21)出願番号　特願2002－92219(P2002－92219)

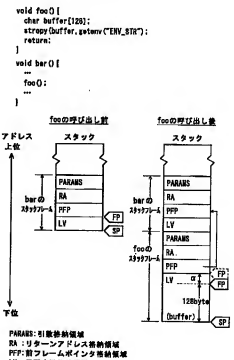
(22)出願日　平成14年３月28日(2002.3.28)

(71)出願人　502110816
有限会社サンアットマーク
京都府京都市北区上賀茂高縄手町116－２
(72)発明者　針川　順行
京都市北区上賀茂高縄手町116－２　有限会社サンアットマーク内
(74)代理人　100095670
弁理士　小林　良平

Ｆターム(参考)　5B017 AA07 CA15
5B033 DE07
5B076 FD00
5B081 CC28

(54)【発明の名称】　メモリ管理方法、メモリ装置、コンピュータシステム、コンパイラ及びプログラム

(57)【要約】
【課題】　スタックスマッシング攻撃を回避するためのメモリ管理技術を提供する。
【解決手段】　サブルーチンスタックの局所変数格納領域LVを確保するに際し、局所変数格納領域LVの大きさを、サブルーチンのソースプログラムで宣言された全局所変数を格納するために必要な大きさより大きい範囲でランダムに決定する。このようにすると、スタックフレーム中でのリターンアドレス格納領域RAの相対位置をソースコード等から事前に正確に知ることは不可能であるため、スタックスマッシング攻撃でリターンアドレスを目的のアドレス（悪意のあるプログラムコードの開始アドレス）に書き換えることは極めて難しくなる。



【特許請求の範囲】

【請求項１】　リターンアドレス格納領域、前フレームポインタ格納領域及び局所変数格納領域を有するスタックフレームを積み重ねて成るサブルーチンスタックをメモリ上に形成する方法において、前記局所変数格納領域の大きさを、サブルーチンのソースプログラムで宣言された全局所変数を格納するために必要な大きさより大きい範囲でランダムに決定することを特徴とするメモリ管理方法。

【請求項２】　請求項１に記載のメモリ管理方法によって形成されたサブルーチンスタック領域を有することを特徴とするメモリ装置。

【請求項３】　請求項２に記載のメモリ装置を備えるコンピュータシステム。

【請求項４】　リターンアドレス格納領域、前フレームポインタ格納領域及び局所変数格納領域を有するスタックフレームを積み重ねて成るサブルーチンスタック領域をメモリ上に形成するための命令をオブジェクトプログラムの内部に埋め込むに際し、その命令を、請求項１に記載のメモリ管理方法によってメモリ上にサブルーチンスタック領域を形成するような命令として生成することを特徴とするコンパイラ。

【請求項５】　請求項４に記載のコンパイラで作成されたプログラム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、スタックスマッシング攻撃により悪意のあるプログラムを実行されるという問題を回避するためのメモリ管理技術に関する。

【０００２】

【従来の技術】インターネットの発達に伴い、悪意のある者が、コンピュータ上で動作しているプログラム（例えば、WWWサーバ、ＦＴＰサーバ）に対してスタックスマッシング攻撃（Stack Smashing Attack）と呼ばれる攻撃を仕掛け、ユーザのＩＤ及びパスワードを盗んでコンピュータに侵入し、データの破壊、改竄、詐取等の不正な活動を行ったり、更には、そのコンピュータを他のコンピュータへの攻撃の踏み台にしたりするという問題が深刻化してきている。

【０００３】スタックスマッシング攻撃では、サブルーチンの実行環境の管理に利用されるスタック型のメモリ領域（以下、サブルーチンスタック又は単にスタックと呼ぶ）を利用して不正なプログラムを実行させる。Ｃ言語で記述されたプログラムを例に、スタックスマッシング攻撃の原理について図２及び図３を参照しながら簡単に説明する。

【０００４】図２にソースコードで示した関数fooは、環境変数"ENV_STR"に格納された値（文字列）をＣ言語の標準ライブラリ関数"getenv"で取得し、その値を同じくＣ言語の標準ライブラリ関数である"strcpy"で局所変数"buffer"に複写する。変数bufferは文字列を格納するための配列変数で、大きさは128バイトである。

【０００５】関数fooを呼び出すとき、プログラムは、fooの実行環境に関する情報を保存するための領域（スタックフレームと呼ばれる）をスタック上に確保する。スタックフレームには次のような情報を確保するための領域が用意される。

・関数fooに渡される引数（Parameters：PARAMS）。

・関数fooを呼び出した関数へのリターンアドレス（Return Address：RA）。

・関数fooを呼び出す前のフレームポインタ。このポインタを本明細書では前フレームポインタ（Previous Frame Pointer：PFP）と呼ぶ。

・関数fooが内部で使用する局所変数（Local Variable：LV）。

【０００６】スタックによるサブルーチン管理ではスタックポインタ（Stack Pointer：SP）とフレームポインタ（Frame Pointer：FP）が用いられる。関数fooが別の関数barから呼び出される場合を例に、上記ポインタの操作を具体的に説明する。なお、ここでは、関数の呼び出しに伴ってスタックがメモリの上位アドレスから下位アドレスに向けて成長するものとする。

【０００７】fooが呼び出される前、スタックポインタSPはbarのスタックフレームの最下位アドレスを指し、フレームポインタFPはbarの前フレームポインタ領域のアドレスを指している。この状態では、フレームポインタFPの指すアドレスより１つ下位のアドレスからスタックポインタSPの指すアドレスまでの領域が、barの局所変数格納領域に相当する。

【０００８】fooが呼び出されると、まずスタックにfooの引数領域Paramsが確保され、そこに引数が格納される。次に、リターンアドレス領域RAが確保され、そこにbarへのリターンアドレスが格納される。次に、前フレームポインタ領域PFPが確保され、そこにその時点におけるフレームポインタFPの値（barの前フレームポインタ領域PFPのアドレス）が格納されるとともに、フレームポインタFPにはfooの前フレームポインタ領域PFPのアドレスが格納される。最後に、fooの局所変数格納領域LVが確保される。なお、スタックポインタSPには、上記のように新たな領域が確保される度に、その領域の最下位アドレスが格納される（つまり、SPは常にスタック全体の最下位アドレスすなわちスタックトップを指している）。以上のような処理の結果、フレームポインタFPの指すアドレスより１つ下位のアドレスからスタックポインタSPの指すアドレスまでの領域が、fooの局所変数格納領域に相当することになり、この状態でfooの処理が実行される。

【０００９】fooの処理が終了したら、上記と逆の手順でフレームポインタFP及びスタックポインタSPの値をfooの呼び出し前の状態に戻す。そして、fooの領域RAに格

納しておいたリターンアドレスにプログラムの制御を戻し、fooの呼び出し後のbarの処理を続行する。

【００１０】fooのスタックフレームの構造は、関数fooのソースコード及びコンパイラの仕様に基づいて容易に解析することができる。この解析結果に基づいたスタックスマッシング攻撃の一例について図３を参照しながら説明する。

【００１１】まず、悪意のあるプログラムのコード３０（Attack Code）を用意する。次に、fooの局所変数格納領域LVからリターンアドレスRAまでカバーするような大きさの文字列３２を用意する。この文字列３２のうち、局所変数格納領域LVに対応する部分にコード３０を書き込んでおき、領域RAに対応する部分にはコード３０の開始アドレスAddr_Xを書き込んでおく。次に、コンピュータの環境変数ENV_STRに何らかの方法で上記文字列を格納する。

【００１２】上記のような状態で関数fooが呼び出されると、foo内部で関数strcpyが実行される。ところで、Ｃ言語の標準ライブラリ関数であるstrcpyは通常、入力文字列が局所変数格納領域LVの大きさより大きいかどうかを自動的に検査するようには実装されていない。従って、図３の場合、文字列３２は局所変数格納領域LVのベースアドレス（フレームポインタFPが指すアドレス）を超えて領域RAまで書き込まれる。こうして、局所変数格納領域LVにコード３０が送り込まれるとともに、リターンアドレス領域RAにはそのコード３０の開始アドレスAddr_Xが格納される。この状態でfooの処理が終了すると、プログラムの制御はbarのリターンアドレスには戻らず、コード３０の開始アドレスに移る。こうして、悪意のあるプログラムが実行されてしまうのである。

【００１３】また、上記攻撃では、strcpyの実行により前フレームポインタPFPの値も書き換えられるため、barへのリターン時にフレームポインタFPを正いいアドレス（barの前フレームポインタPFP）に戻すことができない。更に、文字列３２のうち前フレームポインタPFPに対応する部分に予め計算された不正な値を書き込んでおくことによりプログラムに障害を発生させることも考えられる。

【００１４】スタックスマッシング攻撃に対する対策は従来より各種考えられている。例えば、特開2001-216161号公報に記載の方法や、同公報の中で従来技術として引用されているStackGuardと呼ばれる技術では、攻撃の有無を検査するためにガード値（guard value）を導入している。すなわち、スタックスマッシング攻撃で必ずアクセスされるようなメモリ上の位置にガード値を書き込んでおき、このガード値が関数の実行前後で一致するどうかを関数のリターン処理時に検査する。そして、関数の実行前後でガード値が一致しなかった場合、スタックスマッシング攻撃が行われたものと判断し、エラー処理等を行うのである。また、Ｃ言語に関しては、スタックオーバーフローが発生する可能性がある関数（例えば、strcpy，strcat等の文字列操作関数）を入力文字列の大きさの検査が必ず行われるように実装し直したライブラリ（libsafeと呼ばれる）も提供されている（Arash Baratloo，Timothy Tsai，and Navjot Singh，"Transparent Run-Time Defense Against Stack Smashing Attacks，" in Proceedings of the USENIX Annual Technical Conference，June 2000．http://www.avayalabs.com/project/libsafe/doc/usenix00/paper.html）。

【００１５】

【発明が解決しようとする課題】個人がパーソナルコンピュータで手軽にWWWサーバ等のサーバシステムを構築してインターネット上で公開できるようになってきた今日、スタックスマッシング攻撃により大きな被害が発生する可能性もそれだけ高くなってきている。本発明はこのような問題に鑑みて成されたものであり、その目的とするところは、スタックスマッシング攻撃を回避するためのメモリ管理技術を提供することにある。

【００１６】

【課題を解決するための手段】上記課題を解決するために成された本発明に係るメモリ管理方法は、リターンアドレス格納領域、前フレームポインタ格納領域及び局所変数格納領域を有するスタックフレームを積み重ねて成るサブルーチンスタックをメモリ上に形成する方法において、前記局所変数格納領域の大きさを、サブルーチンのソースプログラムで宣言された全局所変数を格納するために必要な大きさより大きい範囲でランダムに決定することを特徴とする。

【００１７】

【発明の実施の形態及び発明の効果】スタックスマッシング攻撃が可能であるのは、スタックフレームの構造がサブルーチン（Ｃ言語であれば関数）のソースコード及びコンパイラの仕様に基づいて容易に解析することができるからである。このことに着目し、本発明に係るメモリ管理方法では、呼び出されたサブルーチンの局所変数格納領域をサブルーチンスタック上に確保するに際し、前記領域の大きさをコンパイル時に固定的に決定するのではなく、全局所変数を格納するのに必要な大きさより大きい範囲でランダムに決定する。これについて図１を参照しながら以下に具体的に説明する。

【００１８】図１にソースコードで示した関数foo及びbarは図２のものと同じである。いま、関数fooの呼び出しに伴うスタックフレームの形成手順を考える。まず、先に図２を参照しながら説明したのと同様の手順で、引数格納領域PARAMSを確保してそこに引数を格納し、リターンアドレス格納領域RAを確保してそこにbarへのリターンアドレスを格納し、前フレームポインタ格納領域PFPを確保してそこにbarの前フレームポインタ格納領域PFPのアドレスを格納する。

【００１９】次に、局所変数格納領域LVを確保する際、

本発明の方法では、fooの局所変数bufferの大きさ（128バイト）よりも大きい領域を確保する。図１では符号αで示した部分が余分に確保された領域（以下、余剰領域と呼ぶ）に相当する。余剰領域αの大きさは関数の実行時にランダムに決定される。このように局所変数格納領域LVを確保した後、フレームポインタFPに余剰領域αの最下位アドレスを格納する。これにより、フレームポインタFPの指すアドレスをベースアドレスとする局所変数の相対参照が可能になる。

【００２０】上記のような領域確保操作は、前フレームポインタ格納領域PFPと必要な大きさ（図１の例では128バイト）の局所変数格納領域LVとの間にランダムな大きさの余剰領域を別途確保する操作であると解釈することも可能である。

【００２１】なお、上記の方法では、局所変数格納領域LVを確保した後、フレームポインタFPの値を余剰領域αの最下位アドレスに変更するようにしたが、これは本発明にとって必須ではない。例えば、図１に破線で示したように、フレームポインタFPを同領域αの最上位アドレスのまま維持しておき、局所変数へのアクセスに際しては、フレームポインタFPの値に余剰領域αの大きさを加えた値をベースアドレスとする相対参照によりアクセスを行うようにしてもよい。

【００２２】以上のように、本発明に係るメモリ管理方法によれば、同じサブルーチンを呼び出したとしても、局所変数格納領域LVの大きさは呼び出しの度に異なるものとなるため、サブルーチンの実行前に領域LVの大きさを知ることはできない。局所変数格納領域LVの大きさが分からなければ、スタックフレーム中でのリターンアドレス格納領域RAの相対位置をソースコードやコンパイラの仕様に基づいて事前に正確に知ることは不可能である。従って、スタックスマッシング攻撃によりリターンアドレスを目的のアドレス（悪意のあるプログラムコードの開始アドレス）に書き換えることに成功する確率は極めて低くなる。（旧００２３を削除）

【００２３】また、本発明のメモリ管理方法は多様な形態で実装することができる。

【００２４】例えば、本発明の実装形態として、上記メモリ管理方法によって形成されたサブルーチンスタック領域を有することを特徴とするメモリ装置や、そのようなメモリ装置を備えるコンピュータシステムが挙げられる。

【００２５】また、本発明の別の実装形態として、リターンアドレス格納領域、前フレームポインタ格納領域及び局所変数格納領域を有するスタックフレームを積み重ねて成るサブルーチンスタック領域をメモリ上に形成するための命令をオブジェクトプログラムの内部に埋め込むに際し、その命令を、上記メモリ管理方法によってメモリ上にサブルーチンスタック領域を形成するような命令として生成することを特徴とするコンパイラや、そのようなコンパイラで作成されたプログラム（オブジェクトプログラム又は実行可能プログラム）が挙げられる。

【００２６】上記の他にも、本発明はその精神及び範囲内において様々な形態で実装することが可能である。

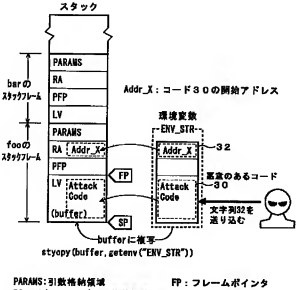
【図面の簡単な説明】

【図１】　本発明におけるサブルーチンスタックの構成例。

【図２】　従来のサブルーチンスタックの構成例。

【図３】　スタックスマッシング攻撃の一例を示す概念図。

【図３】

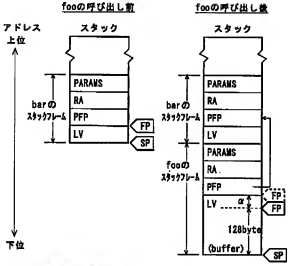


PARAMS：引数格納領域
RA ：リターンアドレス格納領域
PFP：前フレームポインタ格納領域
LV ：局所変数格納領域

FP：フレームポインタ
SP：スタックポインタ

【図１】

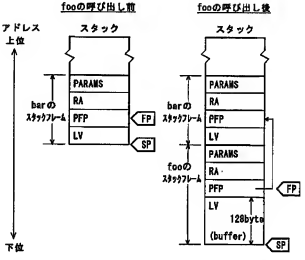
```
void foo(){
  char buffer[128];
  strcpy(buffer,getenv("ENV_STR"));
  return;
}
void bar(){
  ...
  foo();
  ...
}
```



PARAMS:引数格納領域
RA ：リターンアドレス格納領域
PFP:前フレームポインタ格納領域
LV ：局所変数格納領域
FP：フレームポインタ
SP：スタックポインタ

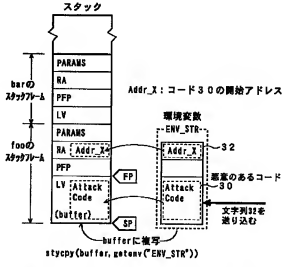
【図２】

```
void foo(){
  char buffer[128];
  strcpy(buffer,getenv("ENV_STR"));
  return;
}
void bar(){
  ...
  foo();
  ...
}
```



PARAMS:引数格納領域
RA ：リターンアドレス格納領域
PFP:前フレームポインタ格納領域
LV ：局所変数格納領域
FP：フレームポインタ
SP：スタックポインタ

【提出日】平成１４年４月１１日（２００２．４．１１）
【手続補正２】
【補正対象書類名】図面
【補正対象項目名】図３
【補正方法】変更
【補正内容】
【図３】



PARAMS:引数格納領域
RA ：リターンアドレス格納領域
PFP:前フレームポインタ格納領域
LV ：局所変数格納領域
FP：フレームポインタ
SP：スタックポインタ